Naval Research Laboratory

Washington, DC 20375-5320

NRL/FR/5510--98-9881

Comparing Simplification Procedures for Decision Trees on an Economics Classification

David W. Aha
Leonard A. Breslow

Navy Center for Applied Research in Artificial Intelligence
Information Technology Division

May 11, 1998

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (*Leave Blank*)	2. REPORT DATE May 11, 1998	3. REPORT TYPE AND DATES COVERED Final

4. TITLE AND SUBTITLE

Comparing Simplification Procedures for Decision Trees on an Economics Classification

5. FUNDING NUMBERS

PN - 55-6470
PE - 62234N
TA - T128

6. AUTHOR(S)

David W. Aha and Leonard A. Breslow

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Research Laboratory
Washington, DC 20375-5320

8. PERFORMING ORGANIZATION REPORT NUMBER

NRL/FR/5510--98-9881

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of Naval Research
800 North Quincy Street
Arlington, VA 22217-5660

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

Several commercial case-based reasoning (CBR) shells now use decision trees to index cases, including ReMind (Cognitive Systems, Inc.), Kate (AcknoSoft), and The Easy Reasoner (The Haley Enterprise). These trees serve to expedite case retrieval and to generate comprehensible explanations of case retrieval behavior. Unfortunately, induced trees are often large and complex, reducing their explanatory power. To combat this problem, some commercial systems contain an option for simplifying decision trees. However, while many methods for simplifying decision trees exist, they have not been systematically compared and most have not been applied to case retrieval. This report builds on our previous survey and initial empirical comparison of tree simplification procedures. In this report, we compare them on a specific, challenging task that is the focus of an existing CBR effort. We examine which tree simplification procedures are useful for this task and suggest which ones should be included in a commercial CBR tool.

14. SUBJECT TERMS

Case-based reasoning
Simplifying decision trees
Indexing
Classification
Decision trees

15. NUMBER OF PAGES

22

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

CONTENTS

COMPARING SIMPLIFICATION PROCEDURES FOR DECISION TREES ON AN ECONOMICS CLASSIFICATION TASK

1. INTRODUCTION

Our research was inspired by a problem that occurred when applying a commercial case-based reasoning (CBR) tool to reason with a large proprietary database on economics. CBR is a popular artificial intelligence problem solving methodology that implements a form of computational analogy, where *cases* are commonly thought of as ⟨problem,solution⟩ pairs. The case-based reasoning problem-solving cycle can be defined by the following four steps (Aamodt and Plaza, 1994):

- *Retrieve*: Given a new problem, retrieve a set of cases from a given *case library* whose problems are *similar* to the new problem.
- *Reuse*: Attempt to apply the solutions of one or more of these stored cases to the new problem.
- *Revise*: Based on feedback from the reuse step, adapt the retrieved solution(s) so that they correctly solve the new problem.
- *Retain*: Store the new problem and its (revised) solution as a new case in the case library.

Commercial off-the-shelf (COTS) CBR tools often represent cases as ⟨feature,value⟩ pairs, where a feature might have either symbolic (e.g., blue, black, red) or numeric values. Most of these tools implement only the *case retrieval* step and are restricted to *classification* tasks, where solutions are simply class labels. Classification involves predicting the class label of a new problem given a library (i.e., data set) of stored cases. An important dependent measure on the quality of a classifier is its *classification accuracy*, the frequency with which its class predictions are correct.

Several COTS CBR tools implement a fast technique for retrieving cases that involves indexing them into a hierarchical data structure called a *decision tree*. Nodes in these trees reference a feature of the cases, while branches are annotated with potential values for the corresponding node's feature. The leaves of these trees typically contain a small number of cases from which a classification decision can be quickly generated or cached. The problem that we address in this report is that, while several CBR COTS tools use decision trees to index cases, they lack procedures to simplify decision trees. Not surprisingly, the trees they generate are large and difficult to comprehend, which is unacceptable to many users.

We surveyed the literature for algorithms that simplify decision trees and created a categorization framework for them (Breslow and Aha, 1997a). We also obtained and installed copies of at least one procedure for each of our framework's subcategories, allowing us to compare them empirically on benchmark classification tasks (Breslow and Aha, 1997b). Our objective was to improve our understanding of the conditions for which each approach is preferred, where the dependent variables were the complexity of the indexing structure (usually a tree) and its classification accuracy. The results were useful because we found several

Manuscript approved November 13, 1997.

expected tradeoffs and a few surprises that motivate further investigation. We can use these results to recommend which simplification procedures to include in a commercial CBR tool and to determine the conditions in which each might prove useful, so that the user can select among them. However, the data sets used in our comparison study were comparatively simple and did not closely resemble those used for commercial, scientific, or government applications. This report addresses this concern. Section 2 briefly describes the complex economics task. In Section 3, we introduce the subject of decision tree induction. Section 4 reviews our framework for categorizing tree simplification procedures and details one example from each of its subcategories. In Section 5, we empirically compare these procedures on the economics data set. We discuss the implications of our results in Section 6, where we also propose which simplification procedures should prove useful in a CBR tool when tackling complex data sets. Section 7 concludes with a summary and our goals for future research.

2. TASK

The economics data set used is a proprietary data set owned by Evidence-Based Research, Inc. We cannot describe some details of this data set nor make it publicly available. However, we can characterize it more generally and report our results.

The tree simplification procedures we surveyed primarily target classification tasks. Although this economics data set can be used for many purposes, this report describes its use in a classification task. Two versions of the economics data set exist: *symbolic* and *numeric* versions. Our study is limited to the symbolic version. Cases in this version are described by 222 features, including a class variable specifying each case's economic classification. There are three classes. All of its features have symbolic values, except for one continuous feature.

The symbolic economics data set contains many *irrelevant* features, defined as features whose values are the same across all cases or are unique to each case. By this definition, there are 28 irrelevant features in this database. We removed them before conducting the empirical study described in Section 5. This reduced the number of features from 222 to 194, including the class variable. Several of the removed features had the value *unknown* for all cases in the data set. After removing these features, the total number of unknown values was reduced from 56,952 (26.3% of the database's values) to 32,552 (18.1%).

Although this preprocessing eliminated several useless variables, there is room for further preprocessing. Several of the remaining features can be encoded more intelligently for use in a CBR tool, but we leave that for future studies.

3. TREE INDUCTION

Section 4 assumes some familiarity with how decision trees are induced. This section introduces a generic *top-down induction of decision trees* (TDIDT) algorithm. Readers familiar with these issues might want to skip this section.

```
Inputs: $C$: Set of training cases
        $I$: Set of index tests
        eval(): Test evaluation function
        stop(): Stopping condition
        post_process(): Post-processing routine
Other Variables:
        $T$: The induced tree
        best(): The best index test at a node as judged by eval()
        $P$: The partition of cases into subsets by applying best($C$)
        $V$: Vector of values returned by best($C$) (one per subset of $P$)
        $N$: A node in $T$

TDIDT($C$,$I$,eval(),stop(),post_process()) =
   $T$ = induce_tree($C$,$I$,eval(),stop())
   return post_process($C$,$I$,$T$)

induce_tree($C$,$I$,eval(),stop()) =
   {best(),$P$,$V$} := best_test_and_partition($C$,$I$,eval())
   IF (stop($P$) = TRUE)
   THEN $N$ := create_leaf_node($C$)
   ELSE $N$ := create_internal_node($C$,best())
        FOR $j := 1$ TO $|V|$ DO
           subtree($N$,$V_j$) := induce_tree($P_j$,$I$,eval(),stop())
   return $N$
```

Fig. 1 — A generic TDIDT algorithm for inducing classification trees

Decision trees are typically induced using a recursive partitioning algorithm on the cases. That is, the set of *training*[1] cases is repeatedly subdivided. The induced tree can classify a test case by traversing from its root to a leaf to yield a classification prediction. Figure 1 displays pseudocode for a generic TDIDT algorithm. These algorithms usually have five inputs:

1. **Training set**: A set of training cases, where each case is typically represented by a set of feature-value pairs plus a class label.
2. **Index tests**: These functions *split* (or *partition*) a group of cases at a node into two or more subsets, where each is represented by a child node. Index tests frequently are simple tests on the value of a feature. An *index* is the combination of an index test and one of its possible values (i.e., represented as a child node).
3. **Test evaluation function**: This evaluates the quality of the splits created by alternative index tests. The objective is to select the test leading to the "best" split (i.e., with highest evaluation). For classi-

[1] We distinguish training and test cases; the former are used to induce the decision tree, while the latter are used to evaluate its performance (e.g., classification accuracy).

fication tasks, an ideal split produces child nodes so that all the cases in a given child are in the same class.

4. **Stopping condition**: This function determines when to discontinue tree expansion.
5. **Post-processing routine**: This routine performs tree-simplification *after* the tree generation process terminates. Post-pruning methods are examples of post-processing routines.

This generic TDIDT algorithm recursively splits the set of training cases into subsets. To select a split, it evaluates the quality of each possible split obtained by applying each index test and selects the "best" one according to the test evaluation function. Tree expansion is terminated whenever the stopping criterion is satisfied. The post-processing routine inputs the tree and attempts to improve it along some measures (e.g., reduce its size).

We use C4.5 Release 7 (C4.5 R7) (Quinlan, 1993), without post-pruning, as the baseline procedure in our experiments. It is a frequently used TDIDT algorithm whose index tests consist of testing only a single feature value.[2] It uses an information theoretic (i.e., gain ratio) test to evaluate the quality of splits. It stops growing the tree when the size of a given set of cases is fewer than a predetermined number.

4. TREE SIMPLIFICATION PROCEDURES

Figure 2 summarizes our framework for tree simplification procedures, which is detailed in Breslow and Aha, 1997a. Each subcategory of the framework is annotated with at least one example tree simplification procedure, namely the one tested in Section 5.[3] The rest of this section summarizes these categories and example algorithms.

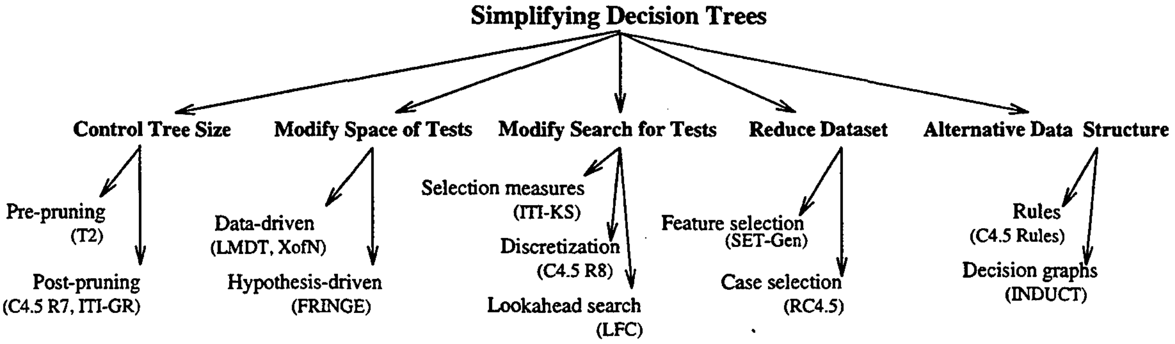

Fig. 2 — A framework for tree-simplification procedures

4.1 Controlling Tree Size

The first of the five categories consists of procedures that explicitly control tree size. It includes two types of *pruning* algorithms. First, *pre-pruners* terminate node expansion unless the estimated quality of the

[2] We're assuming here that C4.5's attribute subset *grouping* option is not being used. (See Section 5.1.1 for more details.)

[3] We later explain that three of these could not be included in our study. (See Section 5 for details.)

best split exceeds a predetermined threshold determined by the stopping condition. Although procedures implementing this approach can be relatively efficient, they tend to *overprune*, terminating tree expansion too early. This can occur when specific combinations of indices (i.e., along paths of the complete tree) have high quality, but they individually do not exceed the predetermined threshold. Nonetheless, we selected a pre-pruner named T2 (Auer et al., 1995) for our experiments because (1) it performed well in comparison studies and (2) it is an extreme foil for our baseline (i.e., C4.5 R7 without post-pruning) in that it limits trees to a depth of 2.

Second, *post-pruners* simplify trees in a post-processing stage. They are intended to increase noise tolerance, prevent overfitting, and eliminate small disjuncts (i.e., leaves with few cases), all in the hope of improving classification accuracy on test sets and yielding more comprehensible trees. We selected C4.5 R7's error-based pruning (EBP) method, which Esposito et al. (1995) found to yield high-accuracy trees, although it tends to *underprune*. We also selected a variant of ITI, ITI-GR (Utgoff, 1995), which uses gain ratio to select tests and the minimum description length (MDL) principle to guide post-pruning. It is an intuitively appealing approach that has not yet been extensively tested. See Mingers (1989) and Esposito et al. (1995) for comparison studies on post-pruning algorithms.

4.2 Modifying the Space of Index Tests

Tree simplification procedures in the second category modify the set of possible index tests. Most early algorithms were limited to *univariate* tests, where each node tests the equality of one feature to one value or an inequality on a numeric feature's value. In contrast, *multivariate* algorithms allow tests on multiple features, multiple values per feature, or both. They target the *subtree replication problem* by clustering cases that univariate tests would otherwise separate. Multivariate algorithms often induce trees with fewer nodes, but multivariate tests are more complex and have higher computational complexity because they search a larger space of index tests.

One way to distinguish multivariate algorithms is by whether they use the previously induced decision tree to generate new combinations of features that can be used in multivariate index tests in the subsequent tree-induction iteration. We call these algorithms *hypothesis-driven* because the tree represents a hypothesis of the true concept description. One such algorithm is FRINGE (Pagallo and Haussler, 1990), variants of which have been shown to solve the subtree replication problem for several benchmark tasks. We obtained a recent version of FRINGE but did not use it because it is limited to binary attributes and binary classification tasks.

We refer to all other multivariate TDIDT algorithms as *data-driven* because they rely solely on the data for guidance. In this subcategory, we selected two promising algorithms. The first is LMDT (Brodley and Utgoff, 1995), which targets numeric features by training perceptrons at each node. The second, XOFN (Zheng, 1995), targets symbolic features. Its tests evaluate conjunctions of an arbitrary set of feature-value pairs. XOFN reduced tree size and increased accuracy, compared with other multivariate approaches that target symbolic features.

4.3 Modifying the Search for Index Tests

The third category of tree simplification procedures includes algorithms that modify how they search for index tests. It consists of three subcategories: algorithms that use an alternative function for evaluating splits, discretize continuous data, or automate lookahead. In the first subcategory, we selected a variant of ITI, named ITI-KS, that differs from ITI-GR only in its use of Kolmogorov-Smirnoff distance rather than gain ratio to evaluate the quality of splits. Utgoff and Clouse (1996) found that it consistently produced smaller trees than did gain ratio, without sacrificing accuracy.

For the second subcategory, we selected C4.5 Release 8 (R8) (Quinlan, 1996), which discretizes continuous features before induction. It uses an MDL approach to penalize continuous features having many values among the training instances, thereby counteracting gain ratio's bias favoring splits on features with many values. Quinlan found that R8 often increased accuracy and reduced tree size relative to R7 for tasks with many continuous features. However, our task has only one continuous feature (out of 194 features), so we cannot expect it to greatly outperform C4.5 R7 here and only investigate whether it substantially reduces speed on this economics prediction task.

For the final subcategory, we selected LFC (Lookahead Feature Construction) (Ragavan et al., 1993). Lookahead procedures evaluate the quality of a given split using information on the quality of subsequent splits. This can be expensive, so LFC constrains lookahead, using a branch-and-bound search. LFC caches the results of lookahead by generating multivariate features, as described in Section 4.2. Ragavan et al. reported that LFC performed well compared with other TDIDT algorithms on a few benchmark data sets, and that both lookahead and constructive induction are needed for it to perform well on tasks involving feature interactions. Unfortunately, we were unable to test it on the economics prediction task because the LFC implementation we obtained is restricted to binary classification tasks.

4.4 Reducing Data Set Size

Methods in the fourth category reduce the database, either through *feature selection* or *case selection*. This has been shown to dramatically reduce tree size. We selected one example from each subcategory. SET-GEN (Cherkauer and Shavlik, 1996) performs feature selection, using a genetic algorithm to search the space of feature subsets. In contrast, Robust C4.5 (RC4.5) (John, 1995) implements an iterative case selection procedure. That is, it induces a tree, post-prunes it, discards cases that are misclassified by it, and iterates unless the new tree correctly classifies all of the remaining cases. Both algorithms performed well compared with C4.5 R7, but have not been compared with one another.

4.5 Alternative Data Structures

Algorithms in the final category transform a decision tree to an alternative data structure. Alternatives include rule sets and graphs. C4.5 RULES (Quinlan, 1993) is a promising simplification method that transforms trees to rules. It tends to greatly reduce the complexity of the concept description and often increases test accuracy (e.g., Peréz and Rendell, 1995). However, Daelemans et al. (1997) reported that C4.5 RULES is prohibitively slow when transforming large (i.e., > 30,000 nodes) decision trees induced by C4.5. We did not expect to have trees of this size in our investigation of the economics data set and thus included C4.5 RULES in our experiments.

Some researchers have also developed promising algorithms for inducing decision graphs. For our experiments, we selected INDUCT (Gaines, 1996), which induces *exception directed acyclic graphs* (EDAGs). EDAGs have several interesting properties. In particular, they relax two key constraints on tree induction. First, they do not require that the union of the cases among a node's children nodes equals the parent's set of cases. Instead, the union need only be a *subset* of the parent's cases, which means that some cases will find their classification predictions at the parent node. Second, a case's feature values may satisfy more than one branch from a node, and these cases must find their classifications by traversing multiple paths. Gaines (1995; 1996) describes a method for translating decision trees to EDAGs and some promising EDAG applications for yielding simpler indexing structures.

5. EMPIRICAL COMPARISON

We compared the selected suite of algorithms on the symbolic economics data set using a 10-fold cross-validation strategy. That is, we (randomly) split the data set into ten equal-sized subsets (*folds*) and, for each fold, tested each algorithm once by using the other nine folds as the training set and the targeted fold as the test set. Thus, all cases are included in the test set on nine of the ten folds, and each algorithm was given identical training and test sets. Our dependent measures were accuracy on the test cases, data structure size, and speed. We tested the algorithms in two modes: the first uses default parameter settings (see Section 5.1) and the second attempts to tune the parameter settings (see Section 5.2).

Data structure size was measured differently for different algorithms. We defined the size of univariate trees as their number of nodes. For multivariate trees, we summed the number of features used at internal nodes with the number of leaves. For C4.5 RULES, we counted the number of rule conditions.

Our intention was to test 13 decision tree simplification algorithms from the 11 subcategories of our framework (see Fig. 2). Unfortunately, we tested only ten. The three not included in our study are FRINGE, LFC, and INDUCT. Our FRINGE implementation is restricted to Boolean features, and we plan to contact other researchers in search of a more robust variant. The LFC implementation we obtained is restricted to binary classification tasks, which prevented us from running it on the ternary classification task defined by the economics data set. Finally, we have, unfortunately, not yet obtained INDUCT even after multiple requests.

5.1 Initial Results

This subsection describes the algorithms' initial results when tested on the economics data set using the 10-fold cross-validation methodology, outlined at the beginning of this section. In this case, they were tested without tuning their parameter values. Instead, their default parameter value settings were used. The Appendix describes these settings in detail.

We divide this section into three subsections, one per dependent measure, and also compare the results to those of C4.5 R7, the benchmark algorithm, without any tree simplification. We write the standard deviation results in parentheses, after specific dependent measure results.

5.1.1 Accuracy

Figure 3 displays the average accuracies for the ten algorithms tested when using only default parameter value settings. The vertical dashed lines distinguish the five categories of our framework (see Fig. 2), and each average accuracy point is shown with a vertical bar denoting its standard deviation. For purposes of brevity, we used the abbreviations "C4R7" ("C4R8") to denote C4.5 R7 (C4.5 R8), "ITI-GR" ("ITI-KS") to denote the variant of ITI using the gain ratio (Kolmogorov-Smirnoff) selection measure, "SGen" to denote

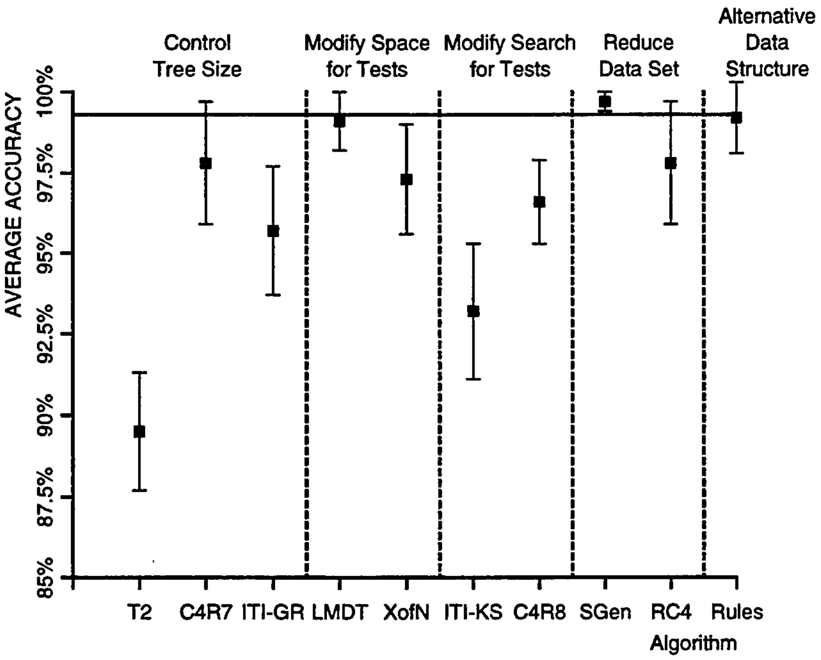

Fig. 3 — Average 10-fold CV accuracies (no parameter tuning)

SET-GEN, "RC4" to denote ROBUST C4.5, and "Rules" to denote C4.5 RULES. The baseline algorithm (C4.5 R7, without pruning) recorded an average accuracy of 99.3% (0.8%), as shown in Fig. 3.

Among the algorithms tested, SET-GEN had the highest average 10-fold CV accuracy on the symbolic economics data set (99.7%). This performance comes at a cost: SET-GEN actually performs a type of parameter tuning (i.e., feature selection). LMDT also performs a similar type of feature tuning but involving feature construction rather than selection. It is difficult, if not unfair, to compare algorithms that tune parameters vs others that do not automatically tune parameters; the accuracies of the former will often be higher, and their elapsed time can be much greater. ROBUST C4.5 also performs a type of parameter tuning in that it performs case selection, and the others also perform various forms of what could be called parameter tuning in their tree simplification procedures but to lesser degrees. Thus, we have strong motivation for repeating this experiment such that *all* the algorithms automatically tune their parameter settings (see Section 5.2).

One reason for SET-GEN'S comparatively high accuracies is that this economics data set probably contains many features that are irrelevant for predicting the target concept. Thus, accuracy improvements might result from discarding these features. Likewise, C4.5 RULES yields accurate classifiers here. We postulate that its transformation of trees to rules allows it to focus on specific feature relationships that are not as easily isolated in a monolithic decision tree. Specifically, where a tree algorithm must either retain or discard all subpaths below a node, a tree-to-rules translation algorithm can selectively retain and then prune entire *paths* in the tree, each path representing a rule.

T2 had the lowest average accuracy (89.5%). It was fooled into selecting symbolic features that have a large number of values per feature. This is a well-known problem with naive index selection algorithms (Fayyad and Irani, 1992), and it has been addressed in several algorithms (e.g., Quinlan, 1993). The key is to remove the index test evaluation function's bias for splits that favor features with the largest number of sym-

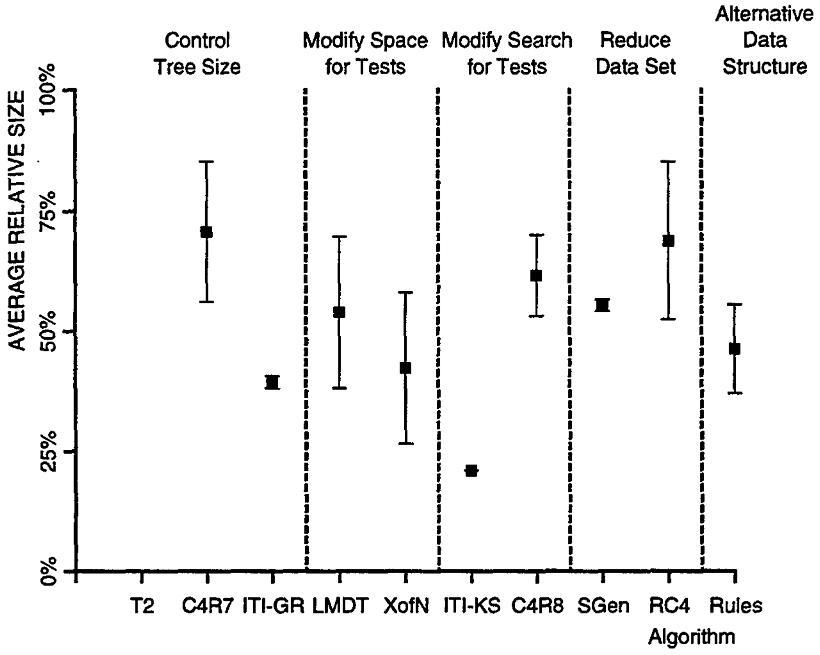

Fig. 4 — Average 10-fold CV data structure sizes (no parameter tuning)

bolic values. (We test a variant of this idea in Section 5.1.2.) However, it appears that this implementation of T2 is highly susceptible, which makes it a poor choice for inclusion in a CBR tool. The ITI-KS variant also performs rather poorly but for a different reason: ITI failed to locate predictive features in general, suggesting that it is struggling to locate good features in this high-dimensional task.

5.1.2 Size

Figure 4 displays the percent average sizes, relative to the baseline algorithm, for nine of the ten algorithms tested, when using only default parameter value settings. The baseline algorithm's (C4.5 R7, without pruning) average size was 160.3 (18.2). The result for T2 is missing because it always generated a tree of size 886, equal to the training set size. This is surprising because T2 was designed to yield small trees. In contrast, the ITI variants do yield small trees, but their zero standard deviations show that they always yield the *same* tree, independent of the fold.

The problem affecting T2's accuracies can be partially eliminated by removing one of its selected features that caused it to perform poorly. That feature, which encodes a date value, is not particularly informative for classification. However, when combined with one other variable that *is* predictive, it uniquely identifies each case in the economics data set. This causes T2 to yield leaves containing only one case. By removing the date feature, T2 induces the same tree across each fold, but it is comparatively smaller (i.e., with "only" 112 nodes) and has perfect test set accuracy. It might prove useful to incorporate a feature selection front-end for T2.

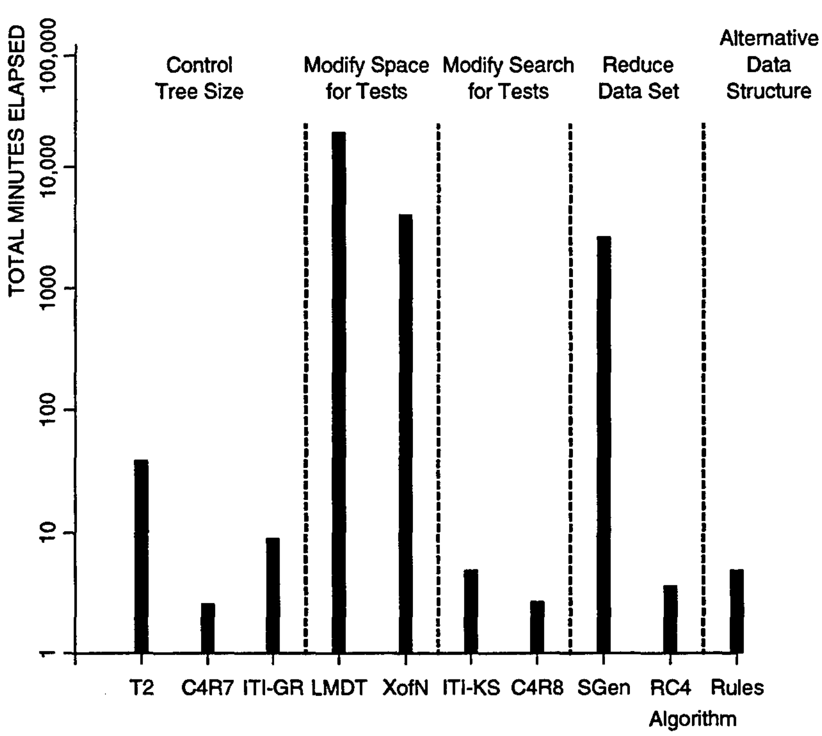

Fig. 5 — Average 10-fold CV computation speeds (no parameter tuning) on a logarithmic scale

5.1.3 Speed

Figure 5 summarizes the real time elapsed, in minutes, for each algorithm to complete the 10-fold CV experiments.[4] The majority of algorithms are reasonably fast, requiring fewer than 10 min to complete all folds for this moderately-sized data set. However, T2 is somewhat slower because it attempts to locate a near-optimal tree of a given size, albeit small in depth. LMDT, XOFN and SET-GEN perform far more expensive searches for feature combinations (LMDT, XOFN) or for feature subsets (SET-GEN) and so are extremely slow on this task (note the log scale in Fig. 5). For example, LMDT required 1713 min (i.e., about 28.5 h) to complete a single fold, when tested on this high-dimensionality task. To achieve practical running times on high-dimensional tasks, the implementations for these algorithms need to be modified to reduce the size of the space they search for feature subsets.

It is instructive to compare these speeds with those recorded, when the algorithms automatically tune their parameters, which is necessary to obtain simpler and more accurate decision trees for a broad range of tasks. We discuss this further in Section 5.1.3.

5.2 Results with Parameter Tuning

This subsection describes the algorithms' results under the same conditions as Section 5.2, except that algorithm parameters are automatically tuned for each fold. Parameter tuning focused on maximizing clas-

[4] These experiments were run on lightly loaded SPARCstation 20's. All implementations are in C. We did not attempt to equate their coding efficiency.

sification accuracy. We explored a few values for each parameter in a factorial design study and selected sets of parameter values that optimized accuracy. Details of this tuning process can be found in the Appendix. We again describe results for three dependent measures: accuracy, data structure size, and speed.

Unfortunately, we report tuning results for only four of the ten algorithms. Because the implementations for ITI and T2 do not have any obvious parameters, they were not included in this second experiment. Furthermore, the current implementations for SET-GEN, XOFN, and LMDT are prohibitively slow for studies involving (additional) tuning of their parameter settings, at least for this economics data set. Thus, these were also not included here.

5.2.1 Accuracy and Size

The remaining algorithms are all variants of C4.5 (Quinlan, 1993). Three of them, namely C4.5 R7, C4.5 R8, and ROBUST C4.5, had highly similar behavior. For a given fold, they all induced the same unpruned and pruned tree.[5] The average size of these trees, when counting only the number of nodes, was 23.8 (3.2) and 19.9 (0.3) nodes, respectively. Each of these induced trees had 100.0% accuracy on their test sets (i.e., across all ten folds). The remaining algorithm, C4.5 RULES, recorded an average accuracy of 99.0% (0.8) and average size of 44.1 (8.2). Thus, tuning produced only most modest gains in its already strong performance.

One of the parameters that was tuned for the C4.5 variants concerns whether the algorithm may split cases by *grouping* values of symbolic features into subsets. This yields a multivariate index test of the form *if this case's value for this feature is in the following subset, then traverse this subtree*. Whereas the default value for this parameter disallows multivariate index tests (*no grouping*), parameter tuning resulted in using multivariate index tests (*grouping*). When using multivariate tests, we should measure the size of the induced trees as we do for the other multivariate algorithms (LMDT, XOFN), taking into account the complexity of the node definitions as well as the number of nodes. Using this definition of size, the three C4.5 decision tree variants all yield trees whose average (pruned) size is 217.9 (or 216.1 for ROBUST C4.5), with a standard deviation of 6.8. These trees are huge. However, some of the features (e.g., date) always selected by these variants could be transformed to a numeric value, which would reduce the tree sizes to an average size of 97.9. This is much closer to, for example, the average size of the trees induced by C4.5 R7's when using the default parameter settings (i.e., 70.7), and intelligent feature engineering could further reduce its size.

In summary, the values selected for cross-validation tuning allowed these C4.5 variants to locate trees with perfect or near-perfect test set accuracy, although parameter tuning did not greatly benefit C4.5 RULES. These results demonstrate how parameter tuning can improve predictive performance by allowing the induction algorithms to explore a much larger space of decision trees than explored when using only the default parameter values. We advocate using automated parameter-tuning methods in any CBR tool that attempts to simplify decision trees.

5.2.2 Speed

Table 1 summarizes the speed, in number of minutes, required to run the 10-fold CV studies of these four tree-simplification procedures. As can be seen, even the fastest of these methods (C4.5 R7) still

[5] Unfortunately, we cannot show example trees and rule sets due to the proprietary nature of the economics data set.

Table 1 — Average 10-Fold CV Speeds (elapsed real-time minutes) (parameter tuning)

Algorithm	Elapsed Time
C4.5 R7	566.1
C4.5 R8	805.3
ROBUST C4.5	1195.8
C4.5 RULES	957.7

required an average of almost one hour per fold when tuning parameters on this data set for the parameter tuning procedures described in the Appendix. This is in stark contrast to the amount of time they required when running only with their default parameter settings, when these algorithms required a sum of barely ten minutes. However, even the slowest elapsed time recorded here, by ROBUST C4.5, is much faster than the three slowest algorithms included in the previous experiment (e.g., where SET-GEN required 2353.2 elapsed minutes).

6. DISCUSSION

Empirical comparisons of tree-simplification procedures have usually involved a small number of tree-simplification procedures because they typically focussed on only *one* category of procedures (see Fig. 2). Those studies, which we surveyed in Breslow and Aha, 1997a, did not compare approaches across all of these categories. In Breslow and Aha (1997b), we compared procedures across all of these categories on eight data sets. However, we chose common benchmark data sets that did not greatly tax the simplification procedures. The purpose of this report is to focus on our sponsor's data set, which inspired our research on procedures for simplifying decision trees. While it presents a more challenging classification task than the earlier benchmark data sets, it is, unfortunately, not available for public disclosure. While we believe our results should extend to data sets with similar characteristics (e.g., high dimensionality, sparse data set), we do not test this hypothesis here and instead leave this for future research.

In the initial experiments reported here, the best-performing algorithm was C4.5 RULES; it recorded high accuracies, had moderately low size requirements, and runs reasonably quickly. These results occurred without parameter tuning, which indicates that C4.5 RULES is a robust algorithm. We are not aware of any commercial CBR tool that uses an explicit rule set to index cases. This seems a promising approach, although C4.5 RULES is known to be prohibitively slow when the induced decision trees are huge (e.g., Daelemans et al., 1997). Transforming trees to graphs for challenging data sets also warrants investigation.

In contrast, algorithms whose search costs are closely tied with the dimensionality of the prediction task are not practical for high-dimensional tasks, such as the economics data set. This includes LMDT, XOFN, and SET-GEN, although some variants of these algorithms might prove useful if they can improve the efficiency with which they perform feature selection.[6] Likewise, T2 needs help when working with high-arity features and could benefit from a good feature selection algorithm. (T2 also struggles when the classification task involves more than four classes (Quinlan, 1996).)

[6] For example, Aha and Bankert (1996) describe how simple variants of popular sequential feature selection algorithms can drastically increase the efficiency of feature selection for a high-dimensional task and can (sometimes) simultaneously improve performance.

Evaluating algorithms using only their default parameter settings often does not provide adequate insight into how well they might perform on a given problem. We tested only four of the algorithms while varying their parameters. Three others have no relevant parameters, while an additional four are so slow that they prevented us from testing them under these conditions. The tuning experiments revealed that the accuracy of the C4.5 variants is enhanced by using multivariate tests on symbolic features, allowing them to locate trees with perfect or near-perfect predictive accuracy. These trees are similar to the one T2 located in the first experiment because both select the same features in the first two levels of their respective trees. However, where T2 generates one branch for each possible value, C4.5 does not, yielding a much smaller tree when counting only the number of nodes. This suggests that T2 could greatly benefit from using multivariate splits, at least under some situations (e.g., when working with highly predictive symbolic features that have many possible values). The improved accuracy resulting from multivariate tests in C4.5 comes at a price of increased tree complexity.

While our previous empirical study (Breslow and Aha, 1997b) focussed on a moderate number of rather simple data sets, this report is complementary because it focuses instead on a more complex data set. Together, they begin to suggest which approaches might be useful for inclusion in a CBR tool, although further studies are warranted (e.g., this study is practically useless for studying C4.5 R8 since its behavior differs from R7 only when using continuous features). In particular, we have some suggestions for CBR tools:

1. *Controlling tree size*: C4.5 variants, which post-prune, are particularly useful due to their relatively fast speeds, although this might simply reflect the maturity of their implementation. We strongly advocate including an efficient post-pruner in a CBR tool. In contrast, we have not yet found evidence showing that pre-pruners are practical tree-simplification approaches.
2. *Multivariate trees*: XOFN, although built on C4.5, is not a practical implementation of approaches that modify the space of indices. In contrast, C4.5's subset *grouping* option for index tests worked well. While multivariate trees have great potential use, care is required in defining the space they explore. Practitioners might want to consider designing multivariate approaches, so that they iteratively explore more constrained spaces of index tests.
3. *Case selection*: Like C4.5 R7, its variant that performs case selection (i.e., ROBUST C4.5) is also relatively efficient and should substantially reduce trees that contain several "small disjuncts" caused by noise.
4. *Feature selection*: This is warranted for practical applications, as demonstrated by SET-GEN's high accuracy in the first experiment. However, efficiency improvements are required before these search-intensive algorithms can solve high-dimensional tasks at practical speeds. We advocate conducting parameter-tuning studies to assist in locating more concise indexing structures.
5. *Alternative data structures*: C4.5 RULES performed particularly well, without parameter tuning, in terms of accuracy, size, and speed (i.e., at least, for the trees induced for this economics prediction task). While we have not yet tested other approaches in this category, such as decision graphs, several published results suggest that they are worthwhile and should be seriously considered for CBR indexing structures.

7. CONCLUSION

This report empirically compared a suite of tree-simplification procedures on a somewhat complex (e.g., high dimensionality) classification task. Our interest was in determining how tree-simplification procedures scale. We found that several of the more elaborate approaches or, at least, their available research implementations, do not scale as well as some of the simpler approaches. Section 6 summarizes our observations and suggestions for which to include in CBR tools. We plan to further evaluate some selected approaches in commercial CBR shells.

This report focuses on an empirical study of tree-simplification tools for classification tasks. Although many tasks can be solved using a classification perspective and attribute-value representations, we are more interested in pursuing synthesis tasks (e.g., problem-solving, planning), and in using more expressive case representations.

Towards these goals, one of our next research objectives is to see how these tree-simplification algorithms compare in the context of graph-structured case representations. In this context, Messmer and Bunke (1995) have recently developed an algorithm that solves a constrained subgraph isomorphism problem in polynomial time, after first generating a decision tree whose size is exponential in the size of the graph-structured cases. Although the authors introduced two methods for pruning their decision trees to combat this problem, these methods either significantly increase retrieval times or do not guarantee subgraph isomorphism detection. We plan to examine whether some of the tree-simplification procedures we studied can reduce tree size without sacrificing polynomial retrieval properties.

Another objective involves studying the use of these tree-simplification procedures for *anytime adaptation* in CBR planning problems, where the environment must be continually monitored to both select goals and to dynamically revise retrieved plans at any point during their execution process. We anticipate studying this in the context of crisis response planning, working closely with members of the Stanford University team who have recently been awarded a large MURI grant for studying this topic. Assuming that crisis response planning tasks benefit from graph-structured case representations, we envision that these two research objectives will merge.

8. ACKNOWLEDGMENTS

Many thanks to Grace Scarborough and Evidence-Based Research, Inc., who collected the economics data, designed the database, and provided it to us for this comparative evaluation. We also thank Patrick Harrison, the leader of NCARAI's Decision Aids Group, who specifically encouraged this work and supported our progress throughout the course of this project. Finally, a thank you to everyone who supplied software and assisted with its use, including Gunnar Blix, Carla Brodley, Kevin Cherkauer, Rob Holte, George John, Ross Quinlan, Jude Shavlik, Paul Utgoff, Ricardo Vilalta, and Zijian Zheng. This research was sponsored by the Office of Naval Research and the Office of Research and Development.

REFERENCES

Aamodt, A. and E. Plaza (1994), "Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches," *AI Communications* 7, 39–59.

Aha, D.W. and R.L. Bankert (1996), "A Comparative Evaluation of Sequential Feature Selection Algorithms," in *Artificial Intelligence and Statistics V*, (Springer-Verlag, New York), pp. 199–206.

Auer, P., R.C. Holte, and W. Maass (1995), "Theory and Applications of Agnostic PAC-Learning with Small Decision Trees," *Proceedings of the Twelfth International Conference on Machine Learning*, Tahoe City, CA, pp. 21–29 (Morgan Kaufmann, San Francisco, CA).

Breslow, L. and D.W. Aha (1997a), "Simplifying Decision Trees: A Survey," *Knowledge Engineering Review* **12**, 1–40.

Breslow, L. and D.W. Aha (1997b), "Comparing Tree-Simplification Procedures," *Proceedings of the Sixth International Workshop on Artificial Intelligence and Statistics*, Ft. Lauderdale, FL, pp. 67–74 (Unpublished).

Brodley, C.E. and P.E. Utgoff (1995), "Multivariate Decision Trees," *Machine Learning* **19**, 45–77.

Cherkauer, K.J. and J.W. Shavlik (1996), "Growing Simpler Decision Trees to Facilitate Knowledge Discovery," *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, Portland, OR, pp. 315–318 (AAAI Press, San Mateo, CA).

Daelemans, W., A. van den Bosch, and T. Weijters (1997), "IGTree: Using Trees for Compression and Classification in Lazy Learning Algorithms," *Artificial Intelligence Review* **11**, 407–423.

Esposito, F., D. Malerba, and G. Semeraro (1993), "Decision Tree Pruning as a Search in the State Space," *Proceedings of the Sixth European Conference on Machine Learning*, Vienna, Austria, pp. 165–184 (Springer-Verlag, Heidelberg, Germany).

Fayyad, U.M. and K.B. Irani (1992), "The Attribute Selection Problem in Decision Tree Generation," *Proceedings of the Tenth National Conference on Artificial Intelligence*, San Jose, CA, pp. 104–110 (AAAI Press, San Mateo, CA).

Gaines, B.R. (1995), "Structured and Unstructured Induction with EDAGs," *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, Montreal, Canada, (AAAI Press, San Mateo, CA).

Gaines, B.R. (1996), "Transforming Rules and Trees into Comprehensible Knowledge Structures," *Advances in Knowledge Discovery and Data Mining* (MIT Press, Cambridge, MA).

John, G. (1995), "Robust Decision Trees: Removing Outliers in Databases," *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, Montreal, Canada, pp. 174–179 (AAAI Press, San Mateo, CA).

Messmer, B.T. and H. Bunke (1995), "Subgraph Isomorphism in Polynomial Time," Technical Report IAM 95-003, University of Bern, Institute of Computer Science and Applied Mathematics, Bern, Switzerland.

Mingers, J. (1989), "An Empirical Comparison of Pruning Methods for Decision Tree Induction," *Machine Learning* **4**, 227–243.

Pagallo, G. and D. Haussler (1990), "Boolean Feature Discovery in Empirical Learning," *Machine Learning* **5**, 71–100.

Pérez, E. and L.A. Rendell (1995), "Using Multidimensional Projection to Find Relations," *Proceedings of the Twelfth International Conference on Machine Learning*, Tahoe City, CA, pp. 447–455 (Morgan Kaufmann, San Francisco, CA).

Quinlan, J.R. (1993), *C4.5: Programs for Machine Learning* (Morgan Kaufmann, San Mateo, CA).

Quinlan, J.R. (1996), "Improved Use of Continuous Attributes in C4.5," *Journal of Artificial Intelligence Research* 4, 77–90.

Ragavan, H. and L. Rendell (1993), "Lookahead Feature Construction for Learning Hard Concepts," *Proceedings of the Tenth International Conference on Machine Learning*, Amherst, MA, pp. 252–259 (Morgan Kaufmann, San Francisco, CA).

Utgoff, P.E. (1996), "Decision Tree Induction Based on Efficient Tree Restructuring," Technical Report 95-18, University of Massachusetts, Department of Computer Science, Amherst, MA.

Utgoff, P.E. and J.A. Clouse (1996), "A Kolmogorov-Smirnoff Metric for Decision Tree Induction," Technical Report 96-3, University of Massachusetts, Department of Computer Science, Amherst, MA.

Zheng, Z. (1995), "Constructing Nominal X-of-N Attributes," *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, Montreal, Canada, pp. 1064–1070 (Morgan Kaufmann, San Francisco, CA).

Appendix

DEFAULT AND TUNED PARAMETER SETTINGS

This Appendix lists the parameter settings used to test the algorithms in the experiments described in Section 5. The order corresponds roughly to the left-to-right ordering shown in the framework displayed in Fig. 2.

The algorithms tested in Section 5 were run in two modes. In the first mode, default settings were used for its parameters. In the second mode, each algorithm's parameter settings were automatically tuned using a 10-fold cross validation process. In this case, we used the entire data set to locate these settings using a *factorial design* on a small number of independent variables (i.e., parameters). That is, we tested the algorithm using all combinations of parameter settings (each combination tested with 10-fold CV), and retained the combination that yielded the highest accuracy. These parameter settings were then used in the 10-fold CV studies whose results are described in Section 5.1.

The following tables describe the default parameter settings (*italicized*), the range of values examined per parameter during the parameter tuning study, and the best parameter settings found while tuning (**boldfaced**). We do not include a boldfaced value for parameters that were not tuned.

Two minor points should be noted. First, the tuned stopping criterion/weight for the three C4.5 variants was 1 for Robust C4.5 and 10 for both C4.5 R7 and C4.5 R8. Second, *no testing* is the system default for C4.5 Rules for the rule condition confidence level parameter, but we used 0.25 because we want to simplify the induced data structures.

Table A1 — Parameter Settings for T2 (Auer et al., 1995) (not tuned)

Parameter	Notation	Value Selected
Tree size	-x	*2-level tree*
Number of splits at bottom nodes	-i x	*Number of classes* $+1$

Table A2 — The C4.5 Variants (R7, R8, Robust) (Quinlan, 1993; 1996; John, 1995)

Parameter	Notation	Values Tested
Pruning confidence level	-c x	{**1**,3,5,10,15,*25*,35,45}
Attribute grouping	-s	{**grouping**, *no grouping*}
Stopping criterion/weight	-m x	{**1**,*2*,3,4,5,**10**,15,20}
Test selection measure	-g	*Gain ratio*
Windowing: number of trees	-t x	*No windowing*

Table A3 — The ITI Variants (Utgoff, 1995; Utgoff and Clouse, 1996) (not tuned)

Parameter	Notation	Value Selected
Error correction mode	-e	*Not used*
Direct metrics	-E -M -N	*Not used*
Non/Incremental mode	-f/-i	*Nonincremental*
Selection measure	-G or -K	*-G* for ITI-GR, *-K* for ITI-KS

Table A4 — LMDT (Brodley and Utgoff, 1995) (not tuned)

Parameter	Notation	Value Selected
Anneal rate	-a	*0.995*
Features elimination	-k	*DSBE elimination*
Test selection measure	-y	*Gain ratio*
Zero weights after each elimination?	-z	*No*

Table A5 — XOFN (Zheng, 1995) (not tuned)

Parameter	Notation	Value Selected
X-of-N's as nominal features	-T x	*When constructed, for tree building*
Pruning confidence level	-c x	*25*
Attribute grouping	-s	*No grouping*
Stopping criterion/weight	-m x	*2*
Test selection measure	-g	*Gain ratio*

Table A6 — SET-GEN (Cherkauer and Shavlik, 1996)

Parameter	Values Tested
Pruning Confidence Levels Per Fold	
Fold 1	.05,.15,*.25*,.35,.45,.55,.65,.75,.85,**.95**
Fold 2	.05,*.15*,.25,.35,.45,.55,.65,.75,.85,**.95**
Fold 3	.05,*.15*,.25,.35,.45,.55,.65,**.75**,.85,.95
Fold 4	.05,*.15*,.25,.35,.45,.55,.65,.75,**.85**,.95
Fold 5	.05,*.15*,.25,.35,.45,.55,.65,.75,**.85**,.95
Fold 6	.05,*.15*,.25,.35,.45,.55,.65,.75,**.85**,.95
Fold 7	.05,.15,*.25*,.35,.45,.55,.65,.75,.85,**.95**
Fold 8	.05,.15,.25,*.35*,.45,.55,.65,.75,**.85**,.95
Fold 9	.05,*.15*,.25,.35,.45,.55,.65,.75,**.85**,.95
Fold 10	.05,*.15*,.25,.35,.45,.55,.65,.75,.85,**.95**
Population size	*100*
Number of children	*4900*
Genome size	*193*
Accuracy weight	*0.75*
Proportion children made by delete-feature	*0.33*
Proportion children made by mutation	*0.33*
Crossover rate	*0.10*
Mutation rate	*0.10*

Table A7 — C4.5 RULES (Quinlan, 1993)

Parameter	Notation	Values Tested
Attribute redundancy ratio	-r	{.8,**.9**,*1.0*,1.1,1.2,1.3,1.5,2.0,2.5,3.5}
Rule condition confidence level	-F	{**No testing**,.01,.05,.10,.15,*.25*}
Pruning confidence level	-c x	{**.01**,.03,.05,.10,.15,*.25*,.35,.45}